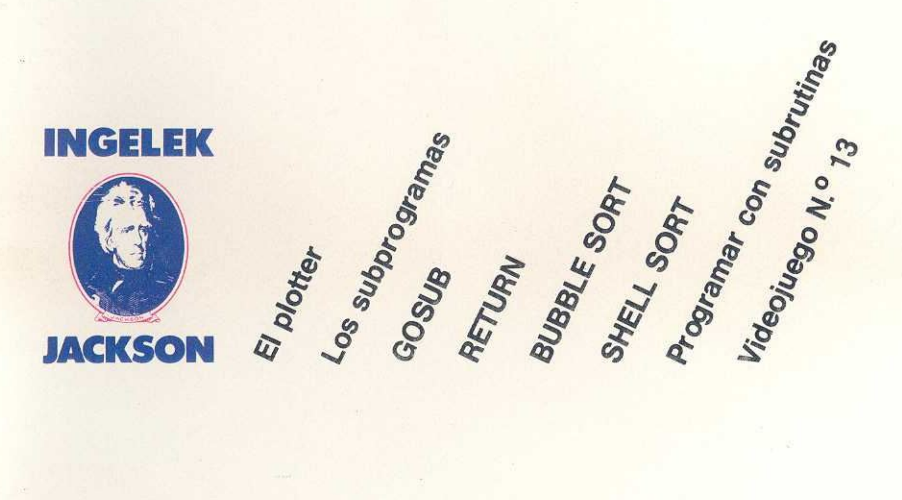


20 LECCIONES DE BASIC
PARA APRENDER CON EL SPECTRUM



13 Spectrum
16K/48K/PLUS

VIDEO BASIC
Una publicación de
INGELEK JACKSON
Director editor por INGELEK:
Antonio M. Ferrer
Director editor por JACKSON HISPANIA:
Lorenzo Bertagnolio
Director de producción:
Vicente Robles
Autor: Softidea
Redacción software italiano:
Francesco Franceschini,
Stefano Cremonesi
Redacción software castellano:
Fernando López, Antonio Carvajal,
Alberto Caffarato, Pilar Manzanera
Diseño gráfico:
Studio Nuovaidea
Ilustraciones:
Cinzia Ferrari, Silvano Scolari,
Equipo Galata
Ediciones INGELEK, S. A.
Dirección, redacción y administración,
números atrasados y suscripciones:
Avda. Alfonso XIII, 141
28016 Madrid. Tel. 2505820
Fotocomposición: Espacio y Punto, S. A.
Imprime: Gráficas Reunidas, S. A.

ISBN del tomo 4: 84-85831-20-9
ISBN del fascículo: 84-85831-11-X
ISBN de la obra completa: 84-85831-10-1
Depósito Legal: M-15076-1985
Plan general de la obra:
20 fascículos y 20 casetes, de aparición quincenal,
coleccionables en 5 estuches.
Distribución en España:
COEDIS, S. A.
Valencia, 245. 08007 Barcelona.
INGELEK JACKSON garantiza la publicación de todos
los fascículos y casetes que componen esta obra y el
suministro de cualquier número atrasado o estuche
mientras dure la publicación y hasta un año después de
terminada.
El editor se reserva el derecho de modificar
el precio de venta del fascículo,
en el transcurso de la obra, si las circunstancias del
mercado así lo exigen.
Octubre, 1985.
Impreso en España.

INGELEK



JACKSON

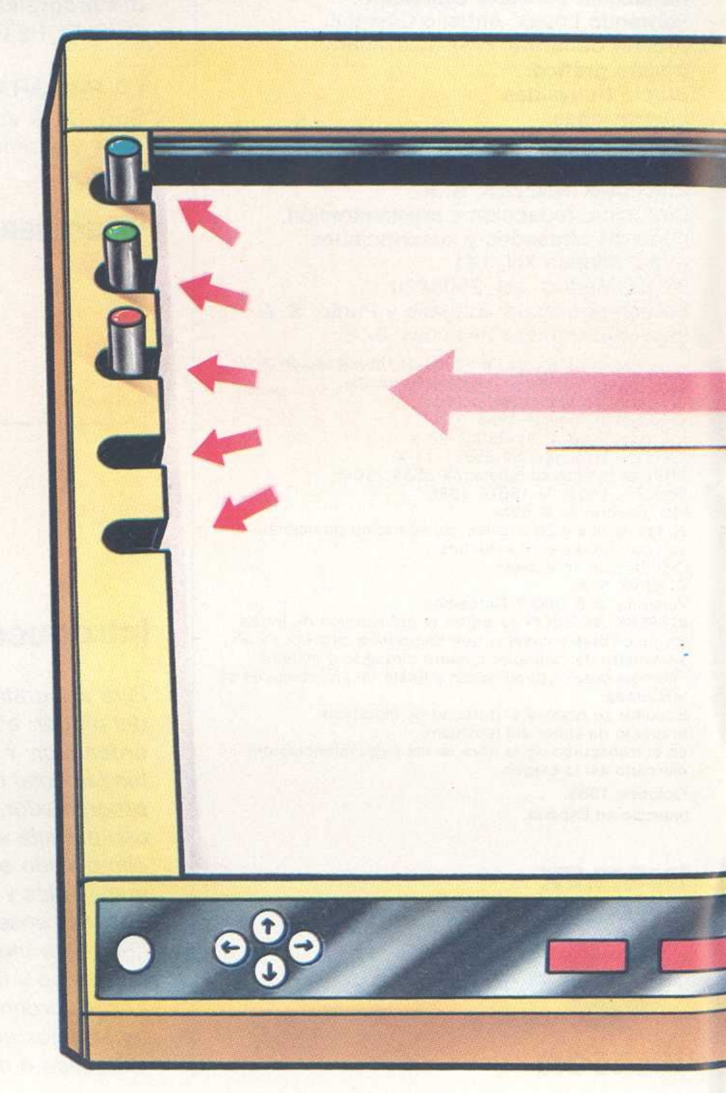
SUMARIO

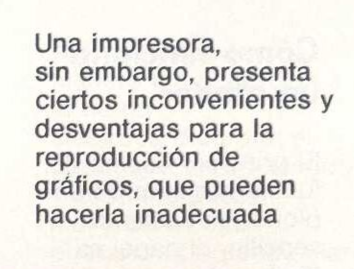
Introducción

Otra vez gráficos, ahora hablamos del plotter: el «tecnígrafo» de tu ordenador. Y además..., un tema fundamental para cualquier programador, los subprogramas. Su uso permite subdividir un programa complicado en pequeñas partes manejables y fáciles de comprender, con las consecuentes ventajas de ahorro de memoria. El mismo ahorro de tiempo y trabajo es ofrecido por el sort (ordenación) que permite tener datos en un cómodo orden alfabético o numérico.

El plotter

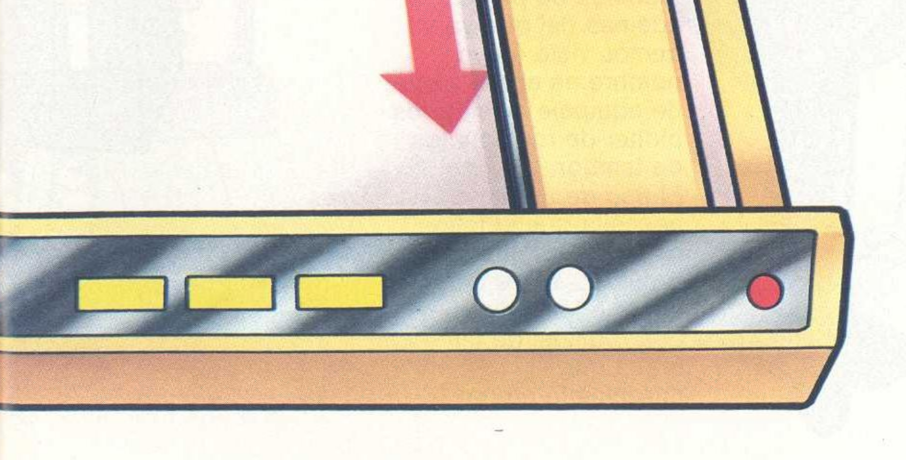
La pantalla constituye el principal periférico de salida de un ordenador personal, pero no el único. Hemos hablado ya de las llamadas impresoras gráficas, periféricos que permiten reproducir sobre el papel —y a veces hasta en colores— todo lo que en un momento determinado aparece en la pantalla.





Una impresora, sin embargo, presenta ciertos inconvenientes y desventajas para la reproducción de gráficos, que pueden hacerla inadecuada para determinados usos, o en circunstancias particulares.

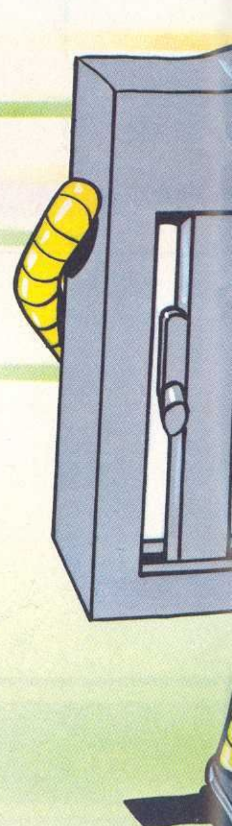
Las imágenes producidas por una impresora no permiten jamás obtener definiciones por encima de un cierto nivel, al estar constituidas por pequeños puntos independientes dispuestos uno junto a otro. Las líneas (ya sean rectas o curvas) están, por lo tanto, siempre



caracterizadas —en mayor o menor grado— por la discontinuidad y desplazamientos de diverso género respecto a la trayectoria ideal que se podría conseguir con un trazo continuo.
Cuando es necesario obtener copias en papel con una mayor definición y precisión, las impresoras gráficas deben ceder su sitio a sus "hermanas mayores", los plotter.
Los plotter son periféricos proyectados y construidos específicamente para dibujar; poseen, por tanto, requisitos y prestaciones gráficas notablemente mejores, de mayor calidad, incluso, que las de la pantalla.

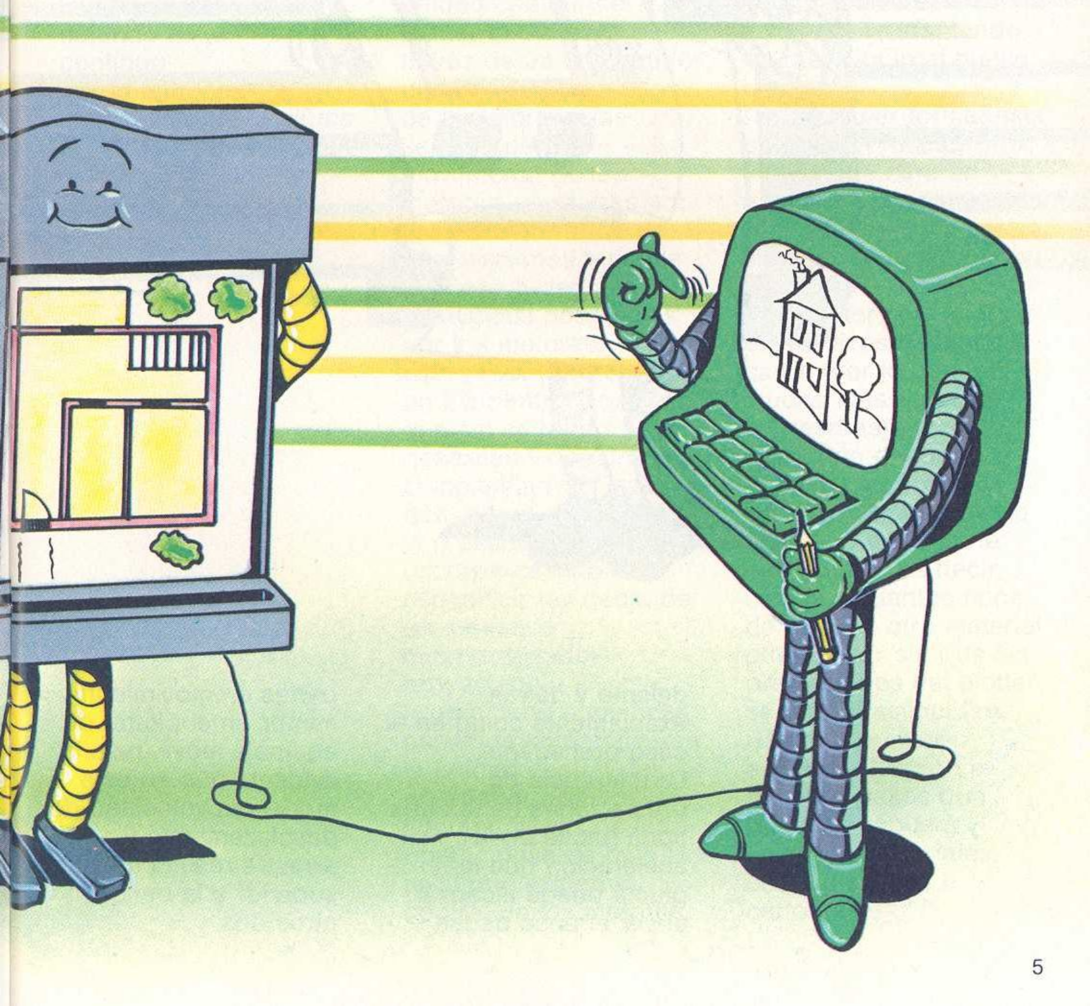
Cómo funciona un plotter

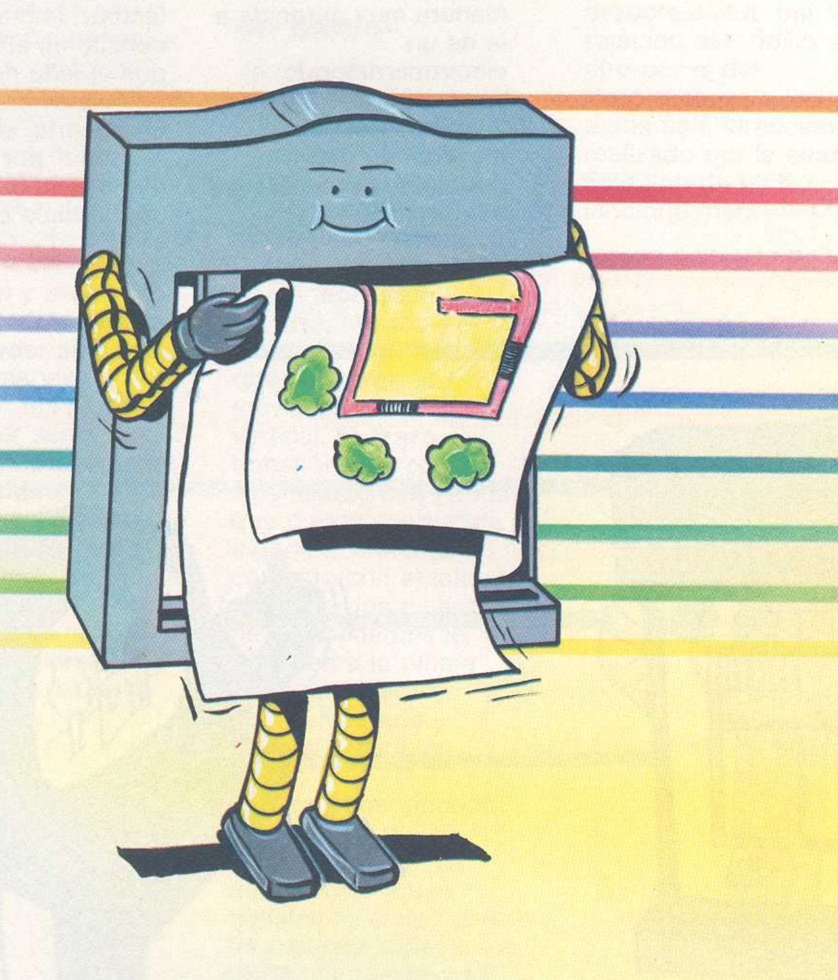
El principio básico del funcionamiento de un plotter es bastante sencillo: el papel es fijado sobre un plano y sobre él se desliza un brazo (muy parecido al de un tecnígrafo normal), movido por dos motores de precisión, que controlan el movimiento horizontal y vertical. El brazo transporta en su movimiento una pluma que puede ser subida o bajada al entrar en contacto con el folio; la suma de dos distintos desplazamientos le permiten a la pluma alcanzar cualquier punto del plano y generar cualquier figura.
Existen otros muchos modelos de plotter, además del que ya hemos visto (cuyo nombre es el de "plotter de equipaje móvil"): los plotter de rodillo y los de tambor.
El tipo de rodillo necesita que el folio esté fijado sobre un rodillo, o sobre un soporte tensado entre dos rodillos, de tal forma que el movimiento en una dirección sea proporcionado por la rotación del rodillo y el otro por el del desplazamiento limitado a una sola dirección, realizado por la pluma.
Este tipo de plotter funciona, para



entendernos, de una manera muy parecida a la de un electrocardiógrafo: el papel gira en un sentido y la pluma puede moverse en otra dirección, perpendicular a la primera.

En cuanto al tipo de tambor, la diferencia consite en el hecho de que el folio no está completamente fijado a un soporte, sino que es "tomado" por un dispositivo de arrastre, que lo hace correr





delante y detrás, exactamente como en el caso precedente.
La diferencia de prestaciones de los tres tipos reside en la aceleración que la pluma puede alcanzar: al ser el peso de las partes en movimiento menor en el plotter de equipaje móvil, resulta evidente que en este modelo la velocidad de desplazamiento y arrastre resulta muy superior a la de los otros dos.

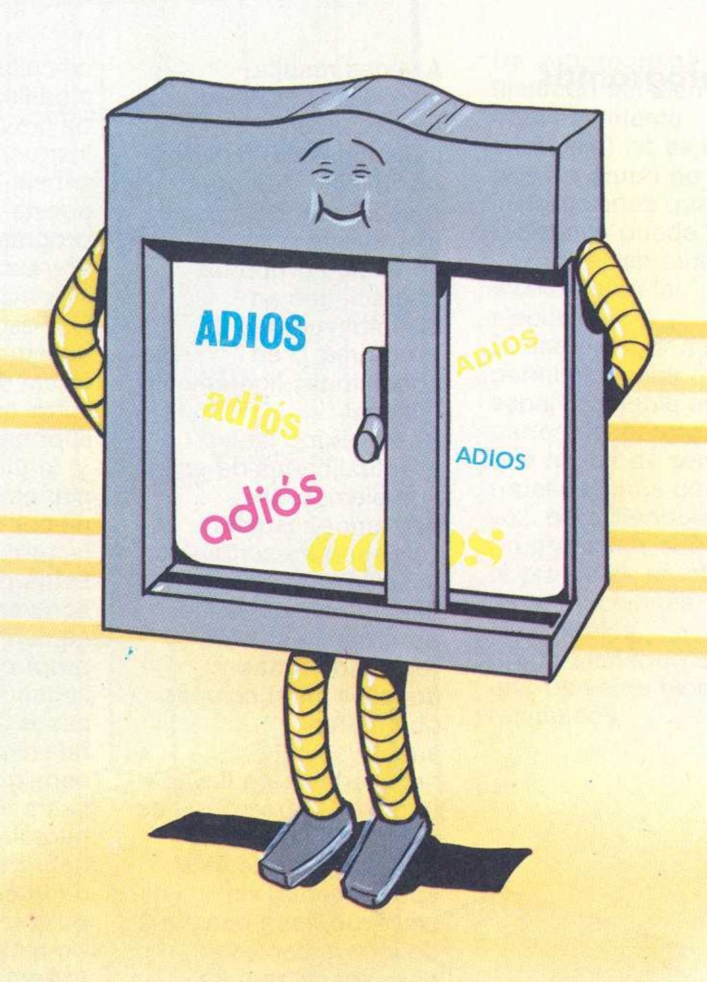
Las líneas trazadas por un plotter, a diferencia de aquéllas representadas sobre una pantalla de pixels o sobre una impresora de agujas, son continuas: la razón de esto se debe a que la pluma —cuando corre sobre el folio dibujando alguna cosa— nunca es levantada del papel, dejando así un trazo continuo
Por lo tanto, el movimiento de la pluma del plotter puede variar continuamente; dicho en otras palabras: el plotter es un dispositivo analógico, aunque ya existen plotter exclusivamente digitales.
Ya que el ordenador solamente es capaz de producir números en forma digital, la conexión entre la unidad central y el plotter debe realizarse a través de un convertidor digital-analógico, capaz de transformar las señales para el movimiento de los motores —enviadas por la CPU bajo forma de informaciones binarias— en magnitudes analógicas aceptables por los motores.
Este problema no existe en los plotter digitales, que sin embargo necesitan —debido a la complejidad del trabajo que deben realizar— de la presencia de un microprocesador específico (es decir, de un auténtico microordenador proyectado y programado a medida), capaz de emparejarse a la unidad central del ordenador principal, liberándola así de las operaciones de bajo nivel que exigen el control de los motores.

La elección de un plotter

Los parámetros que afectan a la elección de un plotter son numerosos. Por ejemplo, el formato de la hoja. Algunos plotter aceptan —si fuera necesario— formatos menores que el estándar, readaptando todo el dibujo al nuevo tamaño.
De cualquier forma, esta prestación resulta inútilmente cara en aquellos casos en que las hojas vayan a ser siempre del mismo tamaño.
Los plotter son dispositivos bastante caros (por lo general, mucho más que las impresoras) y cada accesorio de más aumenta su precio.
Un segundo parámetro es el tipo de soporte que admita, es decir, cuáles y cuántos tipos de papel u otro material puede usar sin que las prestaciones del plotter se vean disminuidas.
Hay que prestar especial atención a aquellos casos que pudieran requerir soportes especiales, como plásticos o cartoncillo.

En cambio, la velocidad es un parámetro que hay que evaluar con referencia a varias consideraciones: cuando se traza una línea a lo largo de una diagonal, la velocidad es mayor que a lo largo de uno de los ejes, al estar compuesta por la suma de dos componentes, horizontal y vertical, que ponen el brazo en movimiento. Además, la mayor parte de los plotter tiene distintas velocidades según la pluma esté levantada o bajada (naturalmente es mayor en el primer caso). Los valores máximos de velocidad para los plotter más comunes van desde menos de 10 centímetros por segundo hasta 1 metro al segundo en los modelos más sofisticados.
Otro parámetro importante es la resolución. Para los plotter se toman en consideración dos tipos de resolución: la mecánica y la direccional. La resolución mecánica viene dada por el mínimo movimiento posible para la pluma, que lógicamente depende de los mecanismos y motores dedicados a esta tarea. La resolución direccionable la da el mínimo movimiento que se puede conseguir con una instrucción vía software.
Por lo tanto, la resolución efectivamente utilizable es la direccionable que como máximo es capaz de igualar a la mecánica.
Otro parámetro interesante es la llamada repetibilidad, que consiste en la capacidad de volver sobre un punto anteriormente tocado, con la menor desviación posible.
Una prueba clásica de repetibilidad consiste en hacer dibujar círculos, figuras planas y rectas paralelas, y hasta efectuando cambios de pluma durante este proceso. Esta última acción es posible porque algunos plotter permiten el cambio automático del color o del grosor de la pluma, tomando la nueva pluma elegida de un portaplumas incorporado, que normalmente se encuentra sobre un borde del plotter, en el cual también se deja la pluma después de cada uso.
Además, en el momento de la compra, pueden resultar necesarias otras prestaciones, que dependen de las aplicaciones específicas a las que vaya a ser destinado el plotter (por ejemplo, ausencia de ruido, sus dimensiones, o el que sea capaz de reconocer instrucciones gráficas específicas).
En su conjunto, un

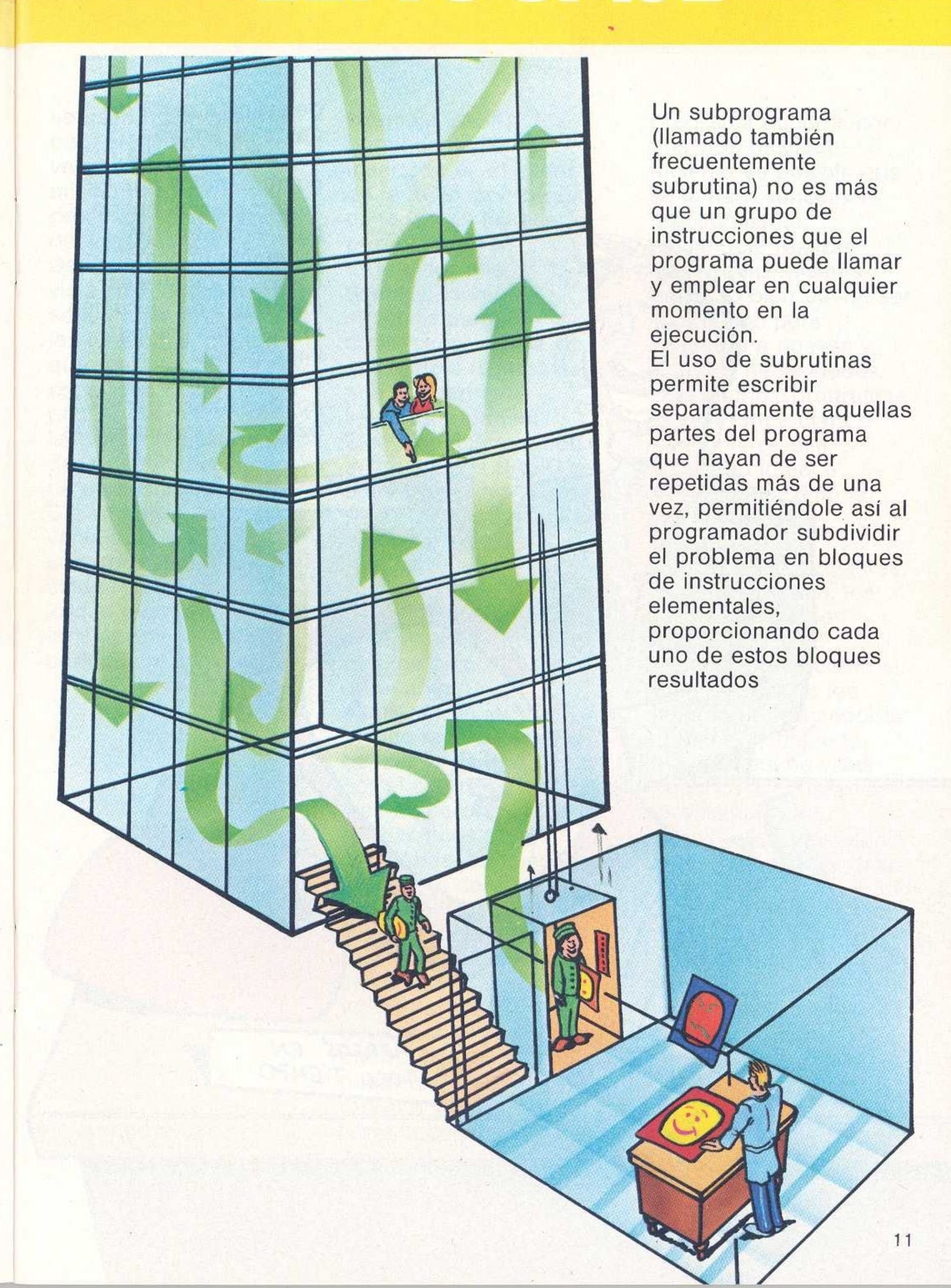
HARDWARE



plotter resulta un periférico hoy día demasiado sofisticado, fundamentalmente apto para usos profesionales y no para diversión. Para un uso normal, y hasta por encima de esto, una "simple" impresora gráfica es seguramente la mejor solución (por lo menos desde el punto de vista económico), al proporcionar en cualquier circunstancia prestaciones más que respetables y poseer una fiabilidad de primer orden.

Subprogramas

A veces resulta extremadamente útil (por no decir indispensable) tener la posibilidad de poder repetir más de una vez una misma operación o un mismo grupo de operaciones en distintos lugares del programa, o en condiciones ligeramente distintas.
Hasta ahora, las eventualidades de este tipo siempre las habíamos resuelto volviendo a escribir las mismas líneas en diferentes puntos del programa, es decir, repitiendo todas aquellas instrucciones que fueran sucesivamente necesarias para llevar a término la tarea que nos habíamos propuesto.
Una solución de este tipo presenta, sin embargo, toda una serie de incovenientes y desventajas que un buen programador no puede dejar de tener en cuenta; veamos estos inconvenientes:
la mayor longitud de un programa (con la ocupación de memoria inútil correspondiente), la mayor posibilidad de cometer errores de tecleado (con sus consiguientes dificultades para la modificación y variación de las diferentes instrucciones en las subsiguientes fases de puesta a punto del programa), la escasa correspondencia entre la forma de razonar a la que estamos acostumbrados y la forma de operar que estas condiciones le imponen al ordenador y, la última pero no menos importante, la menor facilidad de lectura.
Estos problemas se agravan además con el aumento del tamaño del propio programa, llegando en algunos casos a ser tan relevantes que consiguen comprometer hasta trabajos que parecían inicialmente haber sido planteados de la mejor forma posible.
Un método extremadamente eficaz para evitar todos estos incovenientes puede ser el de recurrir al empleo de subprogramas.



Un subprograma (llamado también frecuentemente subrutina) no es más que un grupo de instrucciones que el programa puede llamar y emplear en cualquier momento en la ejecución.
El uso de subrutinas permite escribir separadamente aquellas partes del programa que hayan de ser repetidas más de una vez, permitiéndole así al programador subdividir el problema en bloques de instrucciones elementales, proporcionando cada uno de estos bloques resultados

de salida a partir de datos de entrada.
La parte "sub" del nombre indica que los subprogramas operan bajo el control del programa principal, es decir, en los niveles más bajos de ejecución (más no los menos importantes).
La parte "rutina" del nombre indica por su parte que las subrutinas



se usan habitualmente para ejecutar varias veces dentro de un mismo programa precesos repetitivos, es decir, rutinarios.
Desde este punto de vista, los subprogramas serían comparables a las funciones, puesto que realizan una tarea específica a la que es posible recurrir todas las veces necesarias y desde cualquier punto de un mismo programa.
Sin embargo, la gran ventaja de los subprogramas es que éstos —a diferencia de lo que ocurre con las funciones— no forman parte integrante del lenguaje BASIC, permanentemente insertado en el interior de la ROM, por lo que cada vez es necesario proporcionar al ordenador una definición exacta, especificando, por lo tanto, cada una de las acciones a realizar para que puedan ser llevadas a término.
Esto significa que con instrucciones BASIC es posible escribir subprogramas para resolver cualquier problema, con independencia de las posibilidades o de las prestaciones que ofrezca cada tipo de ordenador.
Así como un libro resulta mucho más fácil de leer (y de modificar por el autor) cuanto más numerosos sean los capítulos que lo componen, también la estructura de un programa puede hacerse más clara y ordenada subdividiendo su cuerpo principal en varias partes, incluidas como subrutinas (cada una de las cuales hasta puede ser escrita y probada con independencia de las demás).
Además, hay que tener en cuenta que un programa puede tener una vida bastante incierta: es posible que se le haga funcionar repetidamente, haciéndole realizar siempre las mismas tareas, o bien puede ser mofidicado para adaptarlo a nuevas y distintas necesidades.
Para escribir programas de una forma clara e inteligible resulta necesario intentar facilitar al máximo la posibilidad de enventuales adaptaciones y mejoras.
En consecuencia, los subprogramas son apreciados universalmente como el arma secreta de los mejores programadores: su uso constituye la mejor tarjeta de visita de un buen programa, permitiendo una presentación más limpia del trabajo, legible, bien organizada y, fundamentalmente, fácil de modificar.
Por lo tanto, intenta aprender, más allá de la pura y simple definición "gramatical", también su funcionalidad y utilidad: habrás alcanzado, con poco pero bien empleado trabajo, una meta importante en el camino de la programación.



GOSUB RETURN

El uso de subprogramas en BASIC es extremadamente sencillo: es suficiente con especificar el número de línea desde la que el subprograma ha de ser iniciado. Diciendo esto así, lo más natural parece recurrir a una simple instrucción GOTO. En cambio, el salto desde el programa principal al subprograma (también

conocido como "llamada" a la subrutina) debe realizarse mediante una breve instrucción especial: GOSUB (abreviatura de GO to SUBrutine).
GOSUB permite transferir la ejecución desde el cuerpo principal del programa al grupo de instrucciones que constituyan un determinado subprograma, de una forma muy semejante a cuanto ocurre cuando se emplea una normal instrucción de salto (GOTO) para transferir el control a otra parte del programa.
La diferencia fundamental e importantísima entre las instrucciones GOSUB y GOTO, es que GOSUB efectúa el salto inicial igual que el GOTO, pero con la ventaja de que recuerda también el punto en el que se inició el salto.
Nos explicaremos mejor: la instrucción

```
30 GOSUB 1000
```

transfiere el control del programa a la línea 1000 (que representa, por lo tanto, el principio de un subprograma), pero memorizando en ese mismo instante –en una zona de la RAM reservada para estos fines– también el número de la línea desde la que impartió la instrucción de salto (es decir, la línea 30).
La ejecución prosigue a partir de la línea 1000 en adelante hasta el momento en que otra instrucción ponga fin a la subrutina. En BASIC esta instrucción es RETURN (cuyo significado –vuelve (retorna)– no deja ningún lugar a la duda).
Cuando el intérprete BASIC encuentra la instrucción RETURN, el control es devuelto a la instrucción inmediatamente siguiente al último GOSUB realizado, en nuestro caso a la línea 30, retomando así la secuencia que el salto a la subrutina habia interrumpido.
Por lo tanto, GOSUB y RETURN siempre han de ser empleadas en pareja: la primera para efectuar la llamada a la subrutina y la segunda para volver al punto inicial.
La falta de la instrucción RETURN al final de un subprograma provoca un considerable jaleo en la memoria del ordenador, con el riesgo de recibir quizás un mensaje de error.
Igualmente peligroso es intentar insertar un RETURN de más: si tu Spectrum encuentra un

RETURN sin haber encontrado antes su correspondiente GOSUB, lo único que puede hacer –dado que ya no sabe desde donde continuar la ejecución– es advertirte del error, visualizando el mensaje.

```
RETURN WITHOUT
GOSUB
```

Si quieres comprobarlo, escribe y ejecuta.

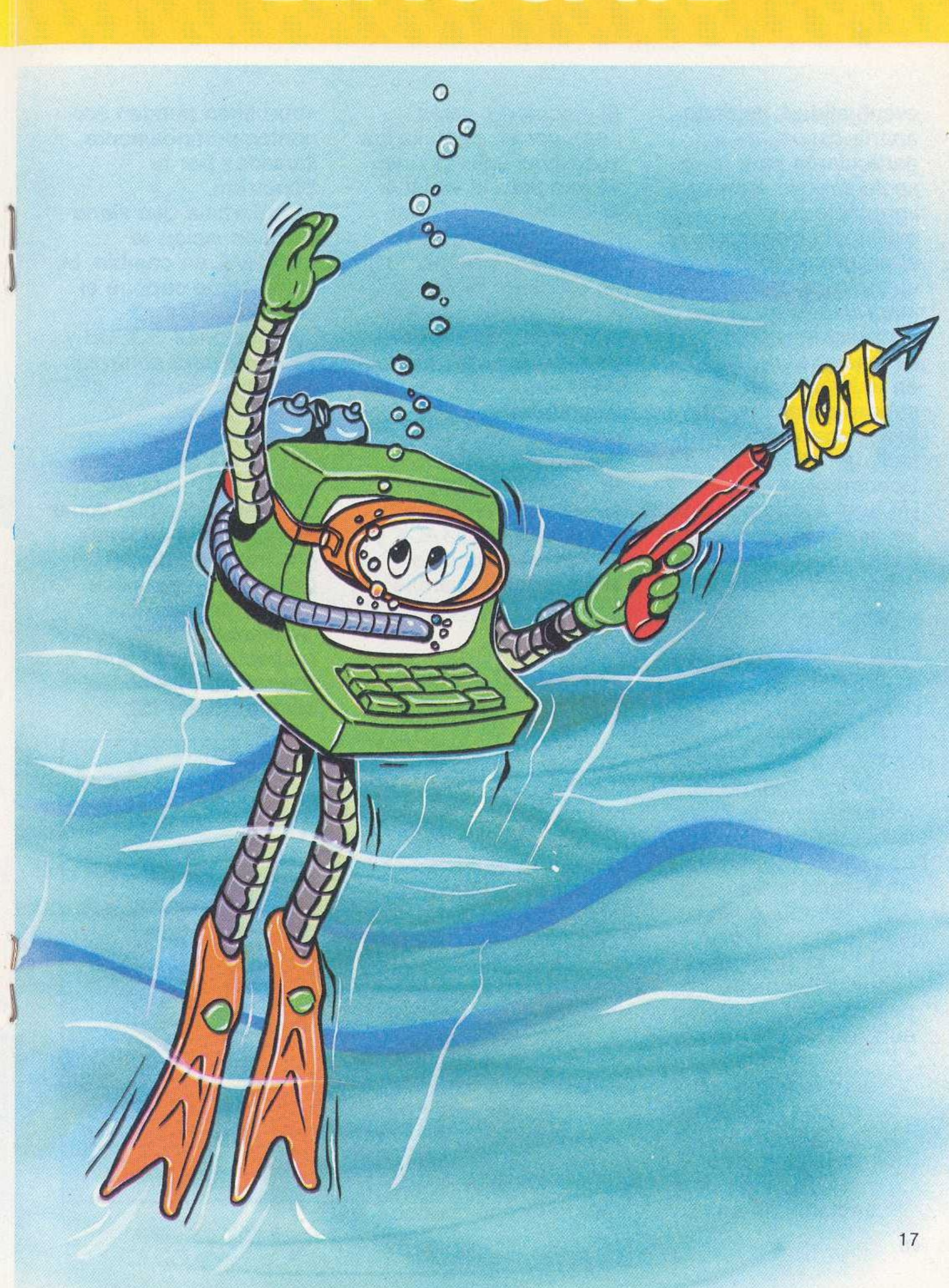
```
10 RETURN
```

y el mensaje de error aparecerá inmediatamente.
De cualquier forma, en un programa pueden encontrar cabida todas las subrutinas que puedas necesitar; será suficiente únicamente con que cada una de ellas disponga de su propio RETURN, que delimitará su final.
Además, en el interior de un subprograma es perfectamente lícito insertar nuevas instrucciones GOSUB, que a su vez también pueden llamar a otras subrutinas, y así sucesivamente. En este caso se habla de subprogramas anidados, es decir, insertados el uno dentro del otro:

```
 10 GOSUB 100
100 ...
110 GOSUB 450
    ...
185 RETURN
450 ...
530 RETURN
```

Sin embargo, es importante no exagerar en el anidado de subprogramas, dado que podría llegar el momento en que el ordenador ya no consiguiera tomar nota de todas las llamadas. Esto es así porque, como ya hemos dicho, la instrucción GOSUB además de realizar un salto hacia el subprograma, memoriza también el número de línea del GOSUB del que parte el salto, en un área de la memoria preparada para ello y llamada STACK (pila). Esta zona se puede imaginar como una pila de páginas superpuestas, en cada una de las cuales se escribe la línea desde la que se debe retomar la ejecución después de cada RETURN.
Con cada GOSUB el intérprete BASIC añade una nueva página a la pila, mientras que con cada RETURN quita una. Pero se puede llegar a un punto en el que ya no haya más páginas disponibles para escribir el número de línea: dicho más aproximadamente, el STACK ha quedado enteramente ocupado. En este caso recibiremos el mensaje de error que nos indica la ausencia de memoria disponible. Esta

LENGUAJE



eventualidad, dejando aparte casos particulares muy raros, suele ser, sin embargo, imputable a la existencia de errores en el programa (por ejemplo, el uso inapropiado de instrucciones GOTO o de bucles FOR...NEXT en el interior de subprogramas puede llenar con inútil facilidad el STACK de tu Spectrum).
He aquí a continuación un (mal) ejemplo, que te ilustra cómo puede sobrevenir este error:

```
10 LET K = 0
20 LET K = K + 1
30 PRINT K
40 GOSUB 20
50 RETURN
```

Al ejecutarlo, verás aparecer en pantalla los sucesivos valores que va tomando la variable K, correspondiente al número de llamadas desde la subrutina 20: en un cierto momento el STACK habrá quedado completamente lleno (dado que el RETURN de la línea 50 nunca es ejecutado) y el ordenador se verá obligado a detenerse.
El último valor presente en pantalla te indicará el tamaño del STACK y en consecuencia te informará de cuantas subrutinas pueden ser contemporáneamente llamadas por tu Spectrum.
El programa que viene a continuación te explicará, en cambio, la ventaja que supone el empleo de las instrucciones GOSUB y RETURN para controlar la gestión de un subprograma.
Supongamos que deseamos que nuestro ordenador —una vez que haya aceptado como entradas dos cadenas cualesquiera de caracteres— visualice tales cadenas «enmarcándolas" respectivamente con asteriscos (*) y con el caracter dólar ($).

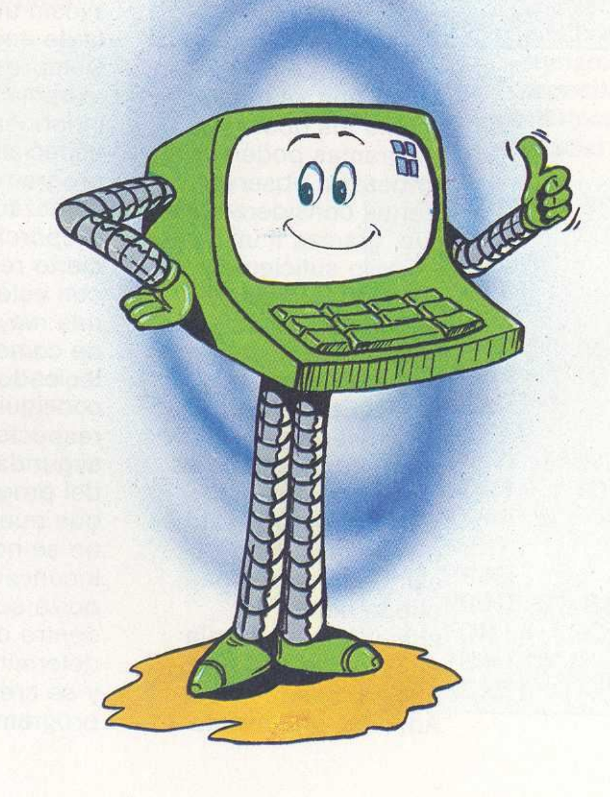
```
10 INPUT A$, B$
20 FOR I = 1 TO LEN (A$) + 4
30 PRINT "*";
40 NEXT I
50 PRINT
60 PRINT "*"; A$; "*"
70 FOR I = 1 TO LEN (A$) + 4
80 PRINT "*";
90 NEXT I
100 PRINT : PRINT
110 FOR I = 1 TO LEN (B$) + 4
120 PRINT "$";
130 NEXT I
140 PRINT
150 PRINT "S"; B$; "$"
160 FOR I = 1 TO LEN (B$) + 4
170 PRINT "$"
180 NEXT I
```

Usando en cambio una subrutina podríamos escribir:

```
 10 INPUT A$, B$
 20 LET C$ = A$ : LET D$ = "*"
 30 GOSUB 500
 40 PRINT D$; "  "; C$; "  "; D$
 50 GOSUB 500
 60 PRINT
 70 LET C$ = B$ : LET B$ = "$"
 80 GOSUB 500
 90 PRINT D$; "  "; C$; "  "; D$
100 GOSUB 500
110 STOP
500 FOR I = 1 TO LEN (C$) + 1
510 PRINT D$;
520 NEXT I
530 PRINT
540 RETURN
```

Ahora que ya hemos escrito los dos programas podemos empezar a observar ciertas consideraciones que, gracias a un ejemplo suficientemente aplicado, es posible evidenciar, y que —con una primera ojeada— pudieran escapar o parecer banales.
En primer lugar el segundo programa es más corto que el primero, pero si esto es a expensas de la claridad y legibilidad puede no ser en absoluto una ventaja (aunque no sea este nuestro caso).
Además, una eventual modificación resulta mucho más difícil realizarla en la primera versión; si, por ejemplo, deseáramos enmarcar la primera cadena con signos de admiración en lugar de asteriscos, sería necesario recorrer todo el programa buscando las instrucciones que hubiera que cambiar (y además, aún a pesar de la facilidad de esta modificación, no dejarían de ser tres las líneas a corregir: la 30, la 60 y la 80).
Y esta última operación, la de modificar, resulta siempre extremadamente larga y laboriosa, especialmente si los programas ya han empezado a tomar unas proporciones de un cierto relieve. Además, con este sistema, existe una mayor probabilidad de cometer errores de tecleado, con la consiguiente desventaja respecto de la seguridad y fiabilidad del programa, puesto que puede ocurrir que no se note el inconveniente (que quizá sólo ocurra dentro de una determinada situación) y se crea que el programa funcione

sin problemas.
En cambio, en el programa con las subrutinas es suficiente con cambiar el signo de admiración por el asterisco en la línea 20 y ya estará todo resuelto.
Otro aspecto a tener en cuenta es la introducción en la segunda versión de dos nuevas variables, C$ y D$. Sus funciones son tan simples como útiles y resultan de uso en multitud de programas.
En muchos casos, un mismo trabajo debe realizarse más veces, pero de una forma ligeramente distinta.
Por ejemplo, en nuestro programa, es necesario imprimir dos filas de caracteres (asteriscos y dólares), basándose en la longitud de dos cadenas distintas (A$ y B$).
Para que la subrutina pueda realizar esta tarea le son necesarias dos informaciones: el carácter a imprimir y la cadena de la que se tiene que calcular su longitud.
Estas informaciones que el programa principal pasa a nuestra subrutina son llamadas "parámetros de entrada" de la subrutina. Puesto que el subprograma en la línea 500 usa como parámetro de entrada las variables C$ y B$, es necesario que el programa principal



antes de llamar a la subrutina vierta a C$ y D$ aquellos valores con los que se haya de operar sucesivamente.
Llegamos finalmente a las dos últimas consideraciones que, aunque sean en apariencia banales, pueden en muchas ocasiones evitar contratiempos y errores aparentemente inexplicables.
La primera hace referencia al lugar de colocación de la subrutina: siempre es conveniente situarla al final del programa principal, numerándola —como en nuestro ejemplo— de tal modo, que salte inmediatamente a la vista. En caso de corrección esto seguramente te evitará errores, dudas, y pérdidas de tiempo.
En cambio, la segunda, hace referencia a la instrucción STOP, que como habrás podido observar, ha sido incluida únicamente en la segunda versión; en la primera ha podido ser excluida —aunque hubiera sido también perfectamente lícito incluirla— puesto que al ser la última instrucción se podía prescindir de ella.
Intenta ahora eliminarla de la segunda versión y ejecuta el programa, verás aparecer en pantalla el error RETURN WITHOUT GOSUB.
Esto es debido a que el Spectrum no sabe donde termina el programa verdadero; en lugar de "comprender" que la ejecución ha de parar después de la línea 100, continúa con las líneas siguientes tratándolas como si formaran parte del cuerpo principal del programa y no de la subrutina.
El RETURN de la línea 540, al no haber sido llamado desde un GOSUB, es el que provoca el error.
La mejor manera de separar el programa principal de la subrutina es la de incluir la instrucción de STOP; de esta forma nos aseguramos evitar posibilidades de errores y de paradas no deseadas.

Sintaxis de la instrucción GOSUB

GOSUB expresión numérica

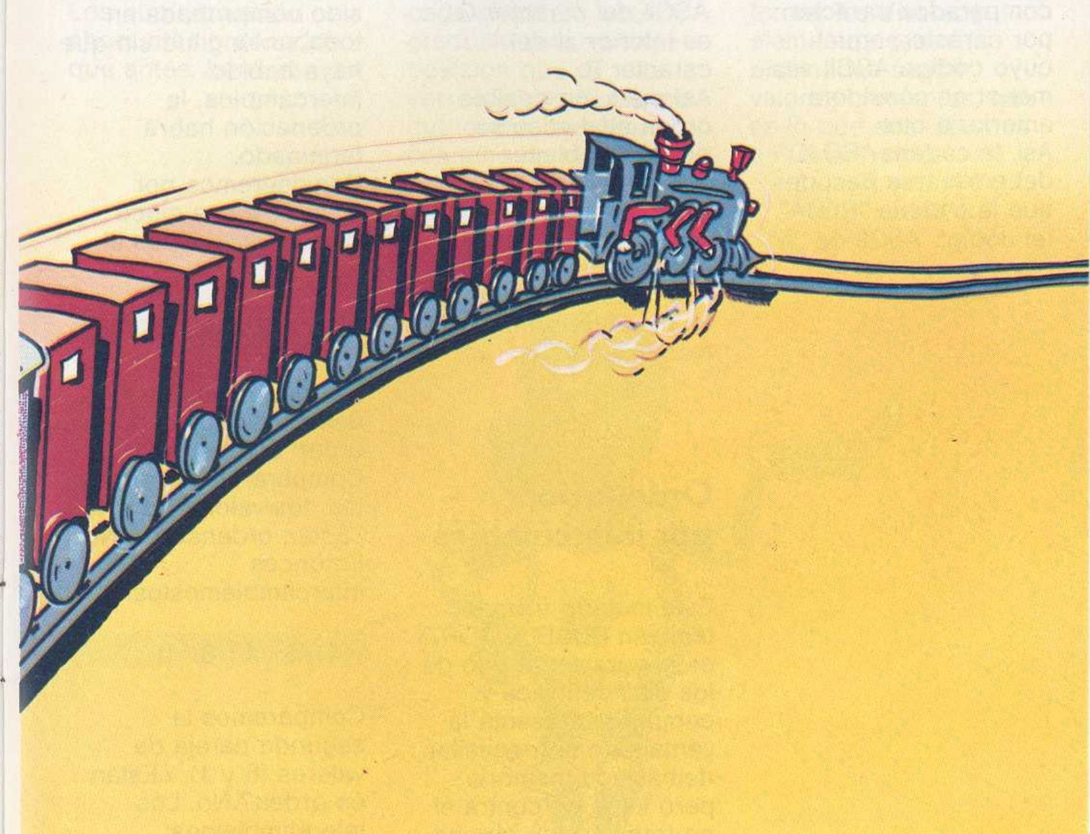
Sintaxis de la instrucción RETURN

RETURN

Sort

Durante la elaboración de datos resulta muy frecuente tener la necesidad de ordenar y clasificar las distintas informaciones (proporcionadas como entradas u obtenidas como resultado de elaboraciones) según determinados criterios o especificaciones.
Normalmente, las informaciones se escriben en el ordenador en un orden más o menos "disperso", mientras que en la salida se necesitan de una forma ordenada según categorias o relaciones claramente precisadas, por ejemplo, alfabéticas o numéricas.
Con la palabra "sort" definimos todas aquellas técnicas de funcionamiento más o menos refinado, cuyo único objeto es el de colocar los datos según un determinado orden.
Cuando se tienen que ordenar centenares o millares de datos, resulta fundamental disponer de algoritmos veloces y eficaces. Con el aumento del número de informaciones a ordenar (que normalmente se insertan en la memoria del ordenador como elementos de cadena) el número de operaciones necesarias

para la ordenación se alarga a veces de una forma excesiva.
Aunque sobre este tema se hayan escrito libros enteros (especialmente adecuados para un uso profesional del ordenador) veremos que, sin embargo, existen algunos términos de ordenación capaces de satisfacer suficientemente las exigencias normales que puede tener un usuario de ordenador personal.
Antes de pasar a ver estos programas será necesario establecer ciertas premisas.
Para efectuar una ordenación, el primer paso necesario consiste en establecer una relación de precedencia entre los elementos a ordenar.
En otras palabras: es necesario establecer qué elemento viene antes y cuál después.
Si para los números no



existen problemas, en cambio el concepto de orden alfabético no resulta tan claro: existen cadenas compuestas por letras, números y símbolos varios. Dejando aparte casos particulares, la ordenación se suele efectuar según el orden de códigos ASCII. Los elementos son comparados carácter por carácter; aquél cuyo código ASCII sea menor se considera anterior a otro. Así, la cadena "ROJO" debe situarse después que la cadena "ROJA" (el código ASCII de "A" es menor que el de "O"), mientras que "RIOJA 1960" antecede a "RIOJA DE HARO" (el código "1" antecede al código "D"). Cuando el ordenador encuentre dos cadenas de caracteres como "GATO" y "RATON" considerará a "GATO" "menor" que "RATON", dado que el código ASCII del carácter G es inferior al del caracter R. Así pues, un posible orden alfabético se podrá establecer utilizando esta característica. Hecha esta claración pasaremos a ver con más detalle dos de las técnicas de ordenación más extendidas.

Ordenación por intercambios

Este método (llamado también BUBBLE SORT) es seguramente uno de los más sencillos y comunes: presenta la ventaja de no necesitar demasiada memoria, pero tiene en contra el no trabajar con mucha velocidad. El método consiste en comparar uno tras otro y por parejas los elementos que compongan una cadena. Si alguna de estas comparaciones constata que los elementos no están ordenados, los intercambia y continúa la comparación con los elementos sucesivos. Cuando la cadena haya sido comprobada en toda su longitud sin que haya habido intercambios, la ordenación habrá terminado. Consideremos por ejemplo estos cinco valores dispuestos al azar:

6 4 1 8 0

y supongamos que deseamos colocarlos en orden creciente. Compararemos primero los dos valores (6 y 4). ¿Están ordenados? No. Entonces intercambiémoslos:

4 6 1 8 0

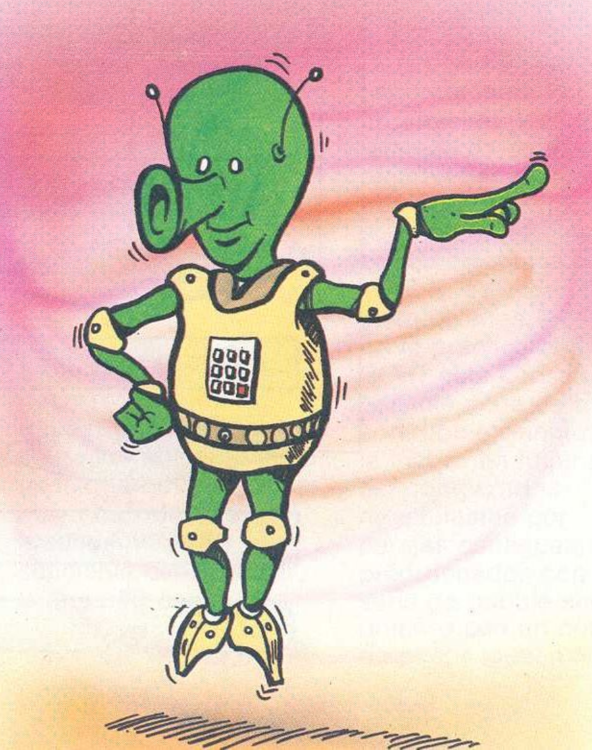
Comparemos la segunda pareja de valores (6 y 1). ¿Están en orden? No. Los intercambiamos:

4 1 6 8 0

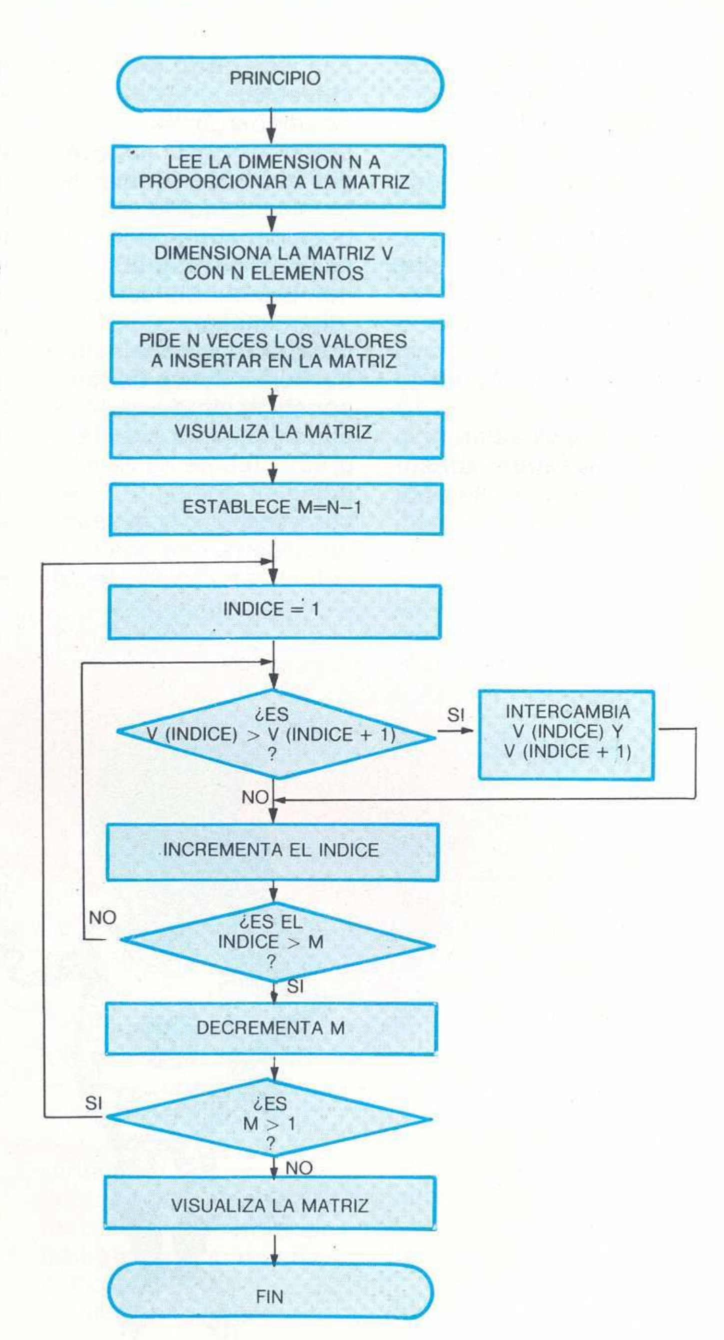
Comparemos la tercera pareja (6 y 8). ¿Están ordenados? Si. Entonces proseguimos manteniéndolos invariables.
Pasemos a la última pareja (8 y 0). Dado que no están ordenados los intercambiamos:

4 1 6 0 8

Los números están ahora más ordenados que antes, pero aún no has alcanzado el orden correcto.
Sin embargo, ya podemos observar que por lo menos el valor 8 ha alcanzado su posición correcta (es decir, la última); el nombre bubble sort —literalmente, ordenación por burbuja— deriva de la constatación de que cada elemento avanza gradualmente hacia la posición que le corresponde, lo mismo que ocurre con una columna de burbujas de aire que se desplace en el agua.
Por lo tanto, podemos volver a comenzar desde el principio todo el proceso, pero teniendo en cuenta que esta vez se puede concluir en la tercera pareja, dado que el último valor ya ocupa su lugar correcto.
Continuando de esta forma, los diversos elementos serán ordenados según valores crecientes, que es lo que deseábamos en un principio.



Puedes ver el diagrama de flujo de todo el proceso en la ilustración de al lado: se incluye también la parte correspondiente al dimensionado de la matriz a ordenar y la de la inserción de los varios elementos, además de la visualización del vector.



Y he aquí el correspondiente listado BASIC:

```
  10 INPUT "CUANTOS ELEMENTOS"; N
  20 DIM V(N)
  30 FOR I=1 TO N
  40 INPUT V(I)
  50 NEXT I
  60 GO SUB 500:REM VISUALIZA LA MATRIZ
     SIN ORDENAR
  70 GO SUB 1000:REM ORDENA LA MATRIZ
  80 GO SUB 500:REM VISUALIZA LA MATRIZ
     ORDENADA
  90 STOP
 500 FOR I = 1 TO N
 510 PRINT V(I); " ";
 520 NEXT I
 530 PRINT:PRINT
 540 RETURN
1000 FOR M=N-1 TO 1 STEP -1
1010 FOR X=1 TO M
1020 IF V(X)>V(X+1) THEN GOTO SUB 2000
1030 NEXT X
1040 NEXT M
1050 RETURN
2000 LET I=V(X)
2010 LET V(X)=V(X+1)
2020 LET V(X+1)=I
2030 RETURN
```

En el caso de que desearas evitarte el trabajo de indicar a mano los números de la matriz (lo que puede ser muy aburrido si son muchos) te será suficiente con sustituir la línea 40 con:

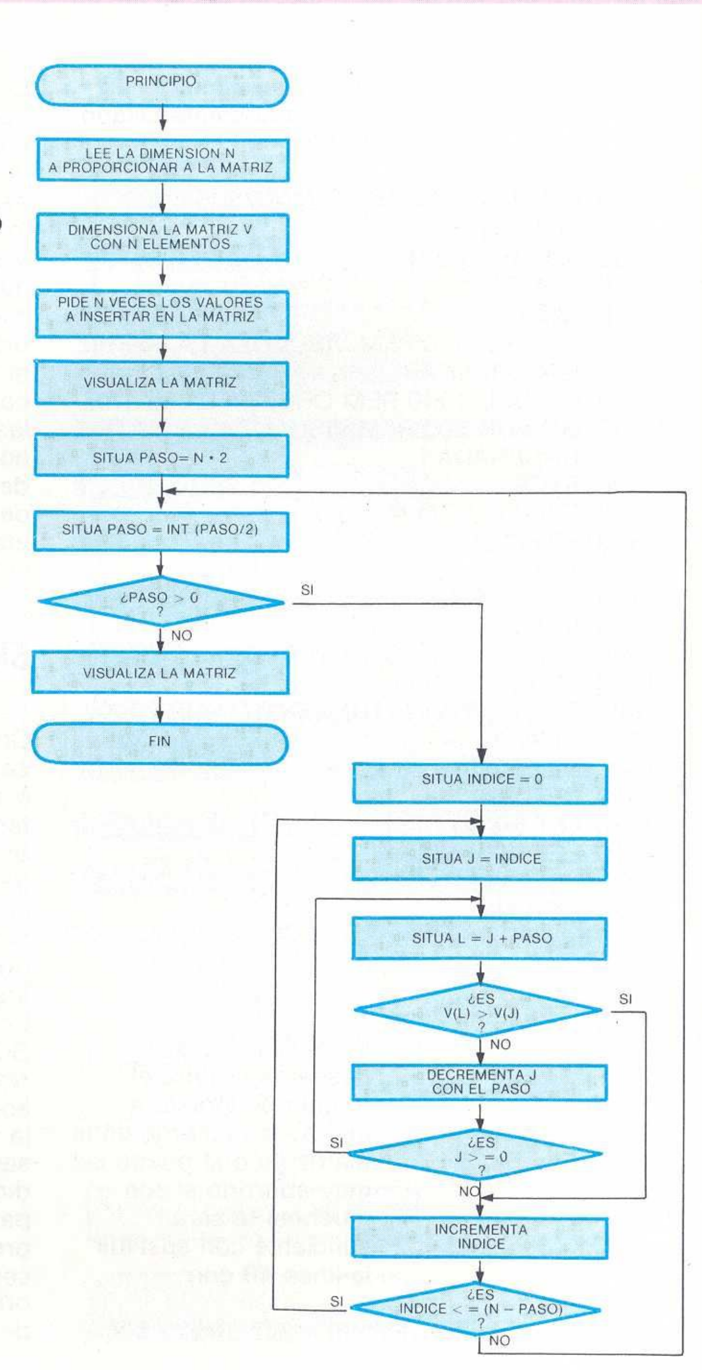
```
40 LET V(I) = INT (RND * 5000)
```

Ejecutando el programa observarás que la ordenación lleva un tiempo más bien largo, especialmente cuando los elementos empiecen a ser bastante numerosos. En efecto, mientras que para la ordenación de 10 datos el programa necesita 45 comparaciones entre las diversas parejas de números, con 250 datos de entrada el número de las comparaciones a efectuar sube hasta ¡31125!

Shell sort

Como podrás ver, este segundo método de ordenación (llamado también de subdivisión binaria) tiene la ventaja de ser más rápido que el anterior, requiriendo prácticamente el mismo número de instrucciones.
En la práctica el SHELL SORT está constituido por una serie de bubble sort. Los elementos de la matriz, en lugar de ser comparados directamente por parejas contiguas, son preordenados con una serie de bubble short: el primero con un número de pasos igual a N

(llamando N al número de elementos de la matriz, es decir, comparando el primer término con el último), el segundo con un paso de N/2, el tercero N/4, y así sucesivamente. Al final, a fuerza de divisiones, se llega al paso 1 y aquí se aplica el bubble sort visto con anterioridad.
He aquí el correspondiente diagrama de flujo:



Y he aquí el listado BASIC:

```
  10 INPUT "CUANTOS ELEMENTOS";N
  20 DIM V(N)
  30 FOR I=1 TO N
  40 INPUT V(I)
  50 NEXT I
  60 GO SUB 500:REM VISUALIZA LA MATRIZ
     SIN ORDENAR
  70 GO SUB 1000:REM ORDENA LA MATRIZ
  80 GO SUB 500:REM VISUALIZA LA MATRIZ
     ORDENADA
  90 STOP
 500 FOR I=1 TO N
 510 PRINT V(I);" ";
 520 NEXT I
 530 PRINT:PRINT
 540 RETURN
1000 LET PASO=2*N
1010 LET PASO=INT(PASO/2)
1020 IF PASO>0 THEN GO SUB TO 1010
1030 RETURN
2000 FOR X=1 TO N-PASO
2020 LET J=X
2030 LET L=J+PASO
2040 IF V(L)>V(J) THEN GO TO 2100
2060 LET M=V(J)
2070 LET V(J)=V(L)
2080 LET V(L)=M
2090 LET J=J-PASO:IF J>=1 THEN GO TO 2030
2100 NEXT X
2110 RETURN
```

Este método necesita aproximadamente 30 comparaciones para ordenar 10 elementos contra los 45 requeridos por el método bubble. En cambio, con 100 elementos son necesarios aproximadamente 900 comparaciones, mientras que el método bubble requeriría 4950. Así pues, sin ser nada especial, la técnica del shell sort resulta muy cómoda en muchas ocasiones, más aún si se considera que el programa requiere una escasa ocupación de memoria.

Aquí también, si desearas evitar insertar manualmente los valores en la matriz, puedes cambiar la línea 40.

```
40 LET V(I)
   = INT (RND*5000)
```

Tu Spectrum sacará automáticamente al azar los números, asignando a continuación estos valores a los elementos de la matriz.

Si en lugar de una matriz numérica deseáramos ordenar una matriz alfanumérica, el programa sería prácticamente idéntico:

```
  10 INPUT "CUANTOS ELEMENTOS"; N
  20 DIM V$(N,4):REM CADA ELEMENTO DE V$
     TIENE 4 ELEMENTOS
  30 FOR I=1 TO N
  40 GO SUB 3000:REM SACA UNA PALABRA
  50 NEXT I
  60 GO SUB 500:REM VISUALIZA LA MATRIZ
     SIN ORDENAR
  70 GO SUB 1000:ORDENA LA MATRIZ
  80 GO SU 500:REM VISUALIZA LA MATRIZ
     ORDENADA
  90 STOP
 500 FOR I=1 TO N
 510 PRINT V$(I);" ";
 520 NEXT I
 530 PRINT:PRINT
 540 RETURN
1000 LET PASO=2*N
1010 LET PASO=INT(PASO/2)
1020 IF PASO>0 THEN GO SUB 2000:GO TO 1010
1030 RETURN
2000 FOR X=1 TO N-PASO
2020 LET J=X
2030 LET L=J+PASO
2040 IF V$(L)>V$(J) THEN GO TO 2100
2060 LET M$=V$(J)
2070 LET V$(J)=V$(L)
2080 LET V$(L)=M$
2090 LET J=J-PASO:IF J>=1 THEN GO TO 2030
2100 NEXT X
2110 RETURN
3000 REM SACA UNA CADENA
3001 REM DA 4 CARACTERES AL AZAR
3005 LET M$=""
3010 FOR L=0 TO 3
3020 LET M=INT(RND*91)
3030 IF M<65 THEN GO TO 3020
3040 LET M$=M$+CHR$(M)
3050 NEXT L
3060 LET V$(I)=M$
3070 RETURN
```

Sólo habríamos tenido que intercambiar el tipo de matriz: alfanumérica en lugar de numérica. Para asignar los elementos de la matriz se ha insertado una subrutina (líneas 3000-3060), que saca cadenas al azar compuestas por 4 carácteres. En el caso de que desearas insertar manualmente los valores de la matriz tendrías que sustituir la línea.

```
40 GO SUB 3000
```

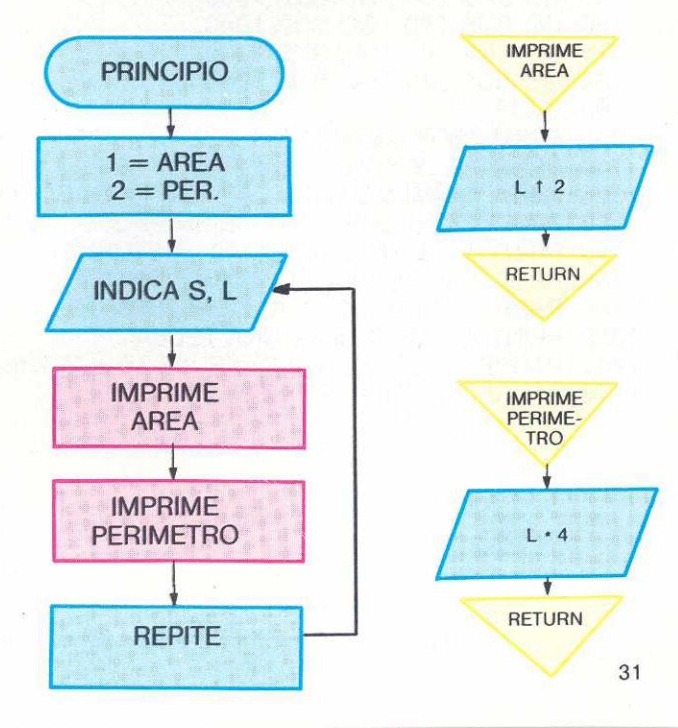
por la línea

```
40 INPUT VET$(I)
```

Area y perímetro de un cuadrado

Este sencillísimo programa demuestra la validez del uso de las subrutinas desde un menú.
Observa en la línea 30 la instrucción GOSUB S * 100. Se trata de un GOSUB "computado": en base al valor de la variable S se llama a una determinada subrutina.

```
10 PRINT "1 = AREA" ' "2 = PER."
20 INPUT "S"; S, "L"; L
30 GO SUB S * 100
40 GOTO 20
100 PRINT L ↑ 2
110 RETURN
200 PRINT L * 4
210 RETURN
```

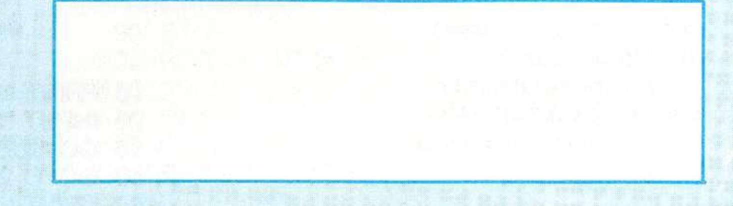


En el Diagrama de Flujo existen 2 nuevos elementos que son característicos de los subprogramas.
Uno, en rojo, identifica la llamada.
El otro, en amarillo, marca el principio y el final de la subrutina.

EJERCICIOS

Anota en los espacios en blanco la SALIDA de los siguientes programas, y compruébalos después con las soluciones de tu ordenador. Si has cometido un solo error tendrás que repasarte la lección.

```
10 CLS : GO SUB 20 : GO SUB 30 : GO SUB 40 : STOP
20 PRINT "¿QUE IMPRIMO EN LA LINEA 40?" : RETURN
30 PAUSE 300 : RETURN
40 RETURN : PRINT "¡¡ESTA NO VAS A PODER VERLA!!"
```



```
  10 CLS
  20 GO SUB 100 : GO SUB 1000
  30 GO SUB 120 : GO SUB 1000
  40 GO SUB 110 : GO SUB 1000
  50 GO SUB 130 : GO SUB 1000
  60 GO SUB 150 : GO SUB 1000
  70 GO SUB 160 : GO SUB 1000
  80 GO SUB 140 : GO SUB 1000
  90 STOP
 100 PRINT "SOY LA PRIMERA QUE SE IMPRIME" : RETURN
 110 PRINT "¡¡¡...Y YO LA TERCERA!!! : RETURN
 120 PRINT " A MI ME HAN LLAMADO LA SEGUNDA..." : RETURN
 130 PRINT " ¡EH! QUE YO TAMBIEN EXISTO" : RETURN
 140 PRINT "¡... Y YO LA ULTIMA!" : RETURN
 150 PRINT "YO SOY LA QUINTA..." : RETURN
 160 PRINT "... YO LA SEXTA..." : RETURN
1000 PRINT AT 3,8; "PULSA UNA TECLA"
1010 PAUSE 0 : LET Z$ = INKEY$ : IF Z$ = "" THEN GO TO 1010
1020 PAUSE 20 : CLS : RETURN
```

